作者：周毅

Rational Suite Enterprise 2002

Purify

使用手册

V 1.0

作者：周毅

EMAIL:foralanzhou@163.com

目录

第一章 功能简介

自动化测试工具 Rational Purify 是 Rational PurifyPlus 工具中的一种,Rational PurifyPlus 包括三种独立的工具：Rational Purify、Rational Purecoverage、Rational Quantify。

Purify是一个面向VC, VB或者Java开发的测试Visual C/C++ 和Java 代码中与内存有关的错误，确保整个应用程序的质量和可靠性。在查找典型的Visual C/C++ 程序中的传统内存访问错误，以及 Java 代码中与垃圾内存收集相关的错误方面，Rational Purify 可以大显身手。Rational Robot 的回归测试与Rational Purify结合使用完成可靠性测试。

只有 Rational Purify 无需源代码或特殊的工作版本，就能检查应用程序代码以及所有链接到该应用程序的构件代码。它可以彻底测试应用程序、检查错误并查明造成错误的特殊构件，从而有助于您得到真实的质量情况， 以便及早纠正。

Java 程序员和测试人员可以将 Rational Purify 和所支持的 JVM 相结合，以改善和优化 Java applet 和应用程序的内存功效。Purify 提供了一套功能强大的内存使用状况分析工具，使您可以找出消耗了过量内存或者保留了不必要对象指针的函数调用。Rational Purify 可以运行 Java applet，类文件或 JAR 文件，支持 JVM 阅读器或 Microsoft Internet Explorer 等容器程序。

使用 Rational Purify 特有的 PowerCheck 功能，可以按模块逐个调整所需的检查级别。这样您就可以把精力集中在最重要的代码上。简单选择“最小”或“准确”即可。“最小”检查可以快速查出常见的运行写入错误和 Windows API 错误；对于关键模块，“准确” 检查将用行业强度检查来查找内存访问错误；这样您就可以确定调试的优先级并更有效地工作。使用 PowerCheck，对每个代码模块指定“最小”或“准确” 的错误检查。对于同时进行代码覆盖分析，请选择覆盖级别，如“代码行”或“函数”，以便更好地控制错误检查和数据覆盖。

在任何 Windows 应用程序中，Windows API 调用都是其重要的组成部分。一个应用程序可能使用成千上万次的 Windows API 调用和 COM 方法。存在内存访问错误的 Windows API 调用，可能会导致应用程序运行不正常或崩溃。对于 Windows API 的检查，Rational Purify 的 WinCheck 功能会验证直到最后一次 Windows API 和 COM 方法的调用情况，包含 GDI 句柄检查和对 Windows 资源泄漏及错误指针等检查。Purify 通过对 API 调用的验证，确保您应用程序的可靠性。

为了使用某些调试工具，您需要经历漫长而乏味的学习过程。一旦使用其中的某个工具，您可能又会发现，该工具并未很好地集成到您的开发环境中。相反，Rational Purify 的学习和使用过程都非常简单。它并不会把您的精力从手头的任务上转移，还能快速找出编程错误。Rational Purify 可以按照您的方式工作，并能弥补您所用工具的不足。由于它是与 Microsoft Visual Studio 集成在一起，所以在您平常工作的地方（Microsoft IDE 中）就可以快速获得 Purify 的自动调试以及源代码编辑功能。这样您在开发流程中遇到的中断将是最少的，同时您的编程热情也丝毫不会受到影响。Purify 带有及时调试功能，当检测到错误时，它将自动停止编程并启动调试器。您也可以通过 Purify 工具栏，将该调试器附加到正在运行的流程中。这将大大增强诊断应用程序中问题的能力，从而缩短查找、复审和修正错误所需的时间。

Rational Purify 可以从多个侧面反映应用程序的质量— 功能、可靠性和性能。通常，质量保证组织只有在进行功能测试过程中偶然碰到了可靠性问题时，才会发觉存在可靠性问题。与内存相关、引起应用程序崩溃的编程错误，并不一定会出现在运行此应用程序的每台计算机上。这些编程错误在开发和测试时可能看不到，只有在最终用户使用此软件时才会显现出来。结果，您只好发布一个又一个的补丁程序来解决这些始料未及的问题。Rational Purify 通过检测影响可靠性的内存相关编程错误，提高 Java 和 C++ 软件的质量。Purify 可在进行功能测试的同时，对可靠性问题进行检测，从而弥补了质量测试的不足。这样就可以为开发人员提供修正问题所需的所有诊断信息。Rational Purify 还能减少错误相互“遮挡”而导致的“测试- 修正”循环的大量时间花费。Purify 主动搜索并记录与内存相关的编程错误，而不是消极地等待应用程序崩溃。它使您可以同时查找多

个错误，并减少软件发布之前所需的“测试- 修正”循环次数。

Rational Purify 是对即将发布的实际C++ 工作版本或在无法获得源代码的情况下进行测试的理想工具。只有 Purify 的专利技术“目标代码插入(Object Code Insertion)”，才无需特殊的工作版本或源代码即可发挥作用。使用Purify，不必为了配合可靠性测试而更改您的构建流程。

第二章 工具特性

2.1 功能描述

2.1.1 可检查的错误类型

1. 堆阵相关错误。
2. 堆栈相关错误。
3. 垃圾内存收集－Java 代码中相关的内存管理问题。
4. COM 相关错误。
5. 指针错误。
6. 内存使用错误。
7. Windows API 相关错误。
8. Windows API 函数参数错误和返回值错误。
9. 句柄错误。

2.1.2 可检测错误的代码

1. ActiveX(OLE/OCX)控件。
2. COM 对象。
3. ODBC 构件。
4. Java 构件、applet、类文件、JAR 文件。
5. Visual C/C++源代码。
6. Visual Basic 应用程序内嵌的 Visual C/C++构件。
7. 第三方和系统 DLL。
8. 支持 com 调用的应用程序中的所有 Visual C/C++构件。

2.1.3 特点

Rational Purify Call Graph 突出显示了泄漏内存最多的 Java 方法。工具提示提供了每种方法的泄漏数据。单击某个方法，即可打开其源代码，以便在编辑器中进行修改。

使用 PowerCheck，对每个代码模块制定“最小”或“准确”的错误检查。对于同时进行代码覆盖分析，选择覆盖级别，如“代码行”或“函数”，一边更好地控制错误检查和数据覆盖。

Rational Purify 会自动找出错误的准确来源和位置。如果有源代码，则可以从 Rational Purify 中启动相应的编辑器，从而快速修复错误。

2.2 参数设置

2.2.1 Settings 项中的 default setting：

1)Error and Leaks：错误和泄漏

Show first message only：在相同的错误第一次出现时显示信息。

Show UMC message：显示 UMC 信息。

Memory leaks：内存泄漏。

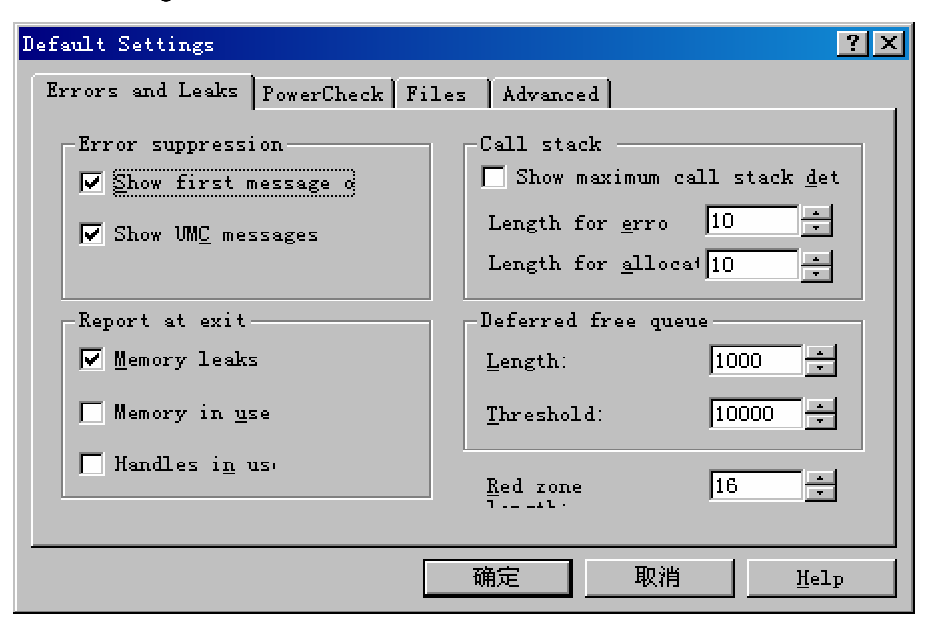
Memory in use：内存使用情况。

Handles in use：句柄使用。

Show maximum call stack detail：最大调用堆栈。

Deferred free queue：延时的自由队列。

Red zone length：亏损区长度。



2)PowerCheck 选项卡：

Default error level：缺省的错误标准。

The module doesn’t contain debugging info：模块无法容纳调试信息。

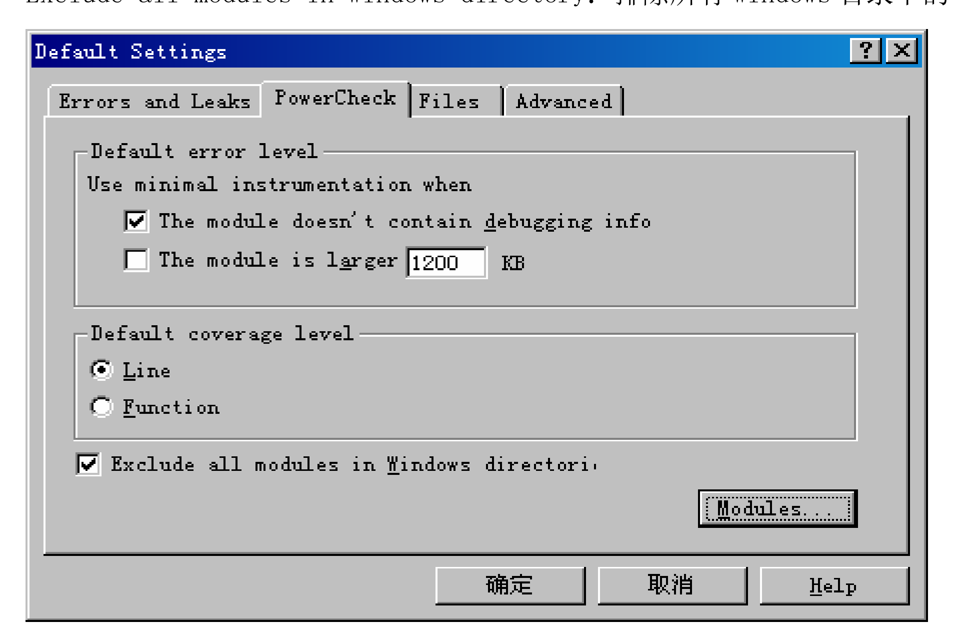
The module is larger……KB：模块大于……字节。

Default coverage level：缺省的覆盖标准。

Line：线程。

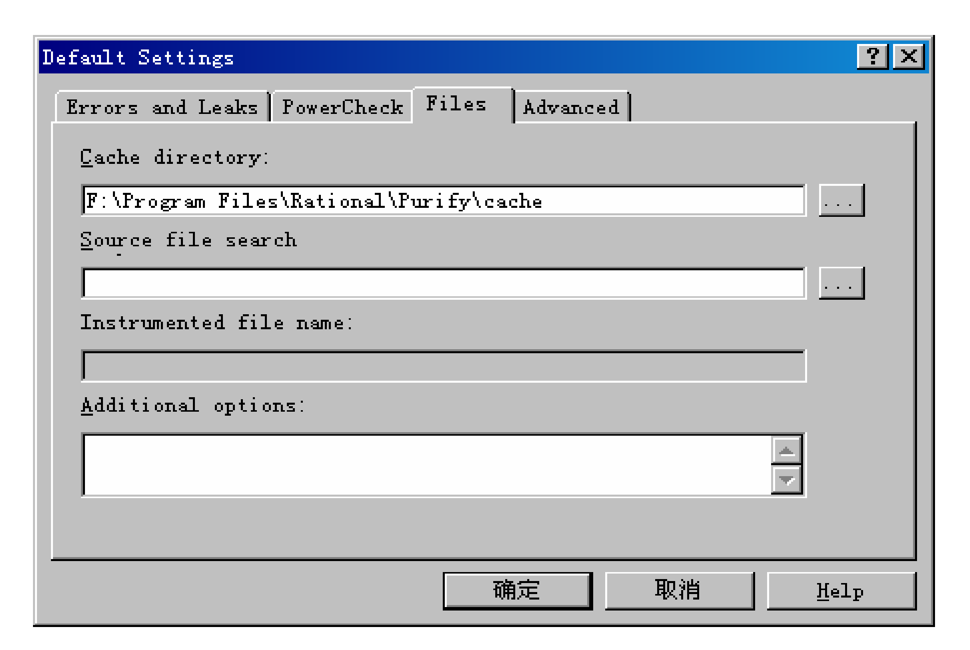
Function：函数。

Exclude all modules in Windows directory：排除所有 Windows 目录下的模块。



3)Files 选项卡：

在此选项卡中设置相关文件的路径及填写附加信息



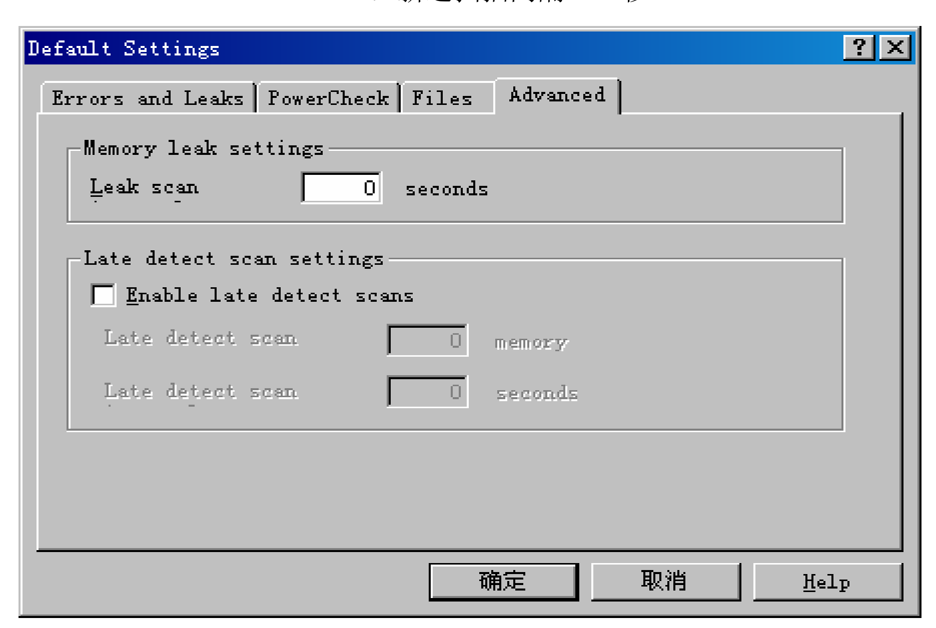
4)Advanced 选项卡：

Leak scan interval：泄漏扫描间隔。

Enable late detact scans：能够察觉新近的扫描。

Late detact scan：新近扫描……堆。

Late detact scan interval：新近扫描间隔……秒。



2.2.2 Settings 项中的 Preferences：

1)Runs 选项卡：

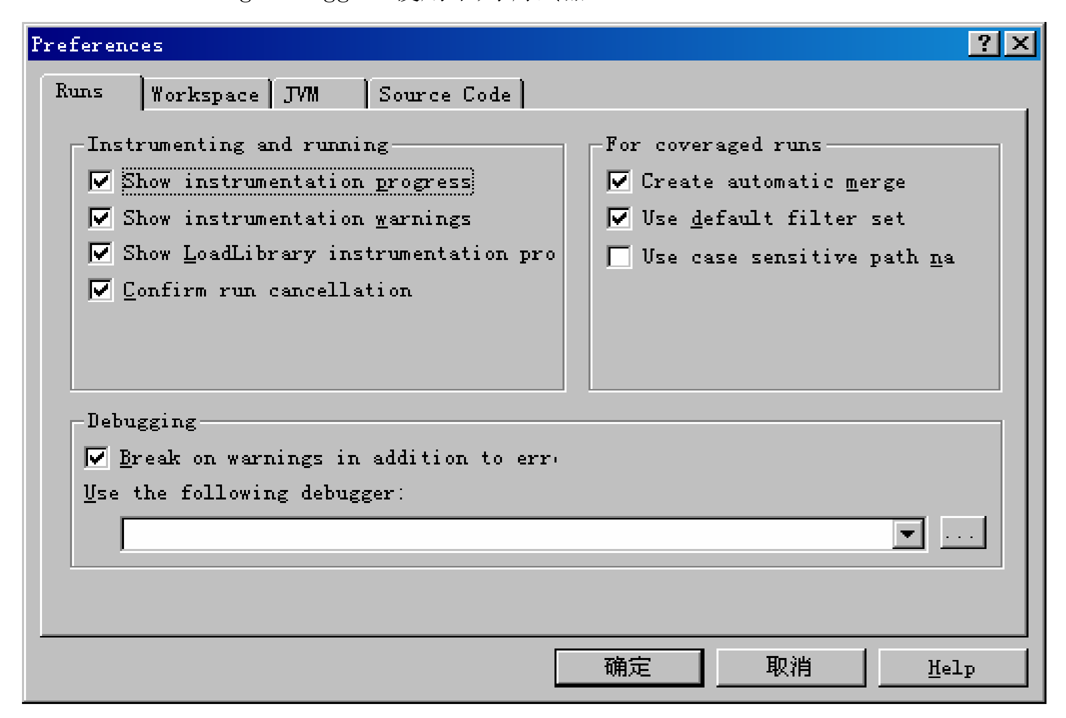
Show instrumention progess：当对 VC++程序测试时，是否显示工具对话框。

Show instrumention warnings：当对 VC++程序测试时，侦探到警告对话框，是否显示工具警告对话框。

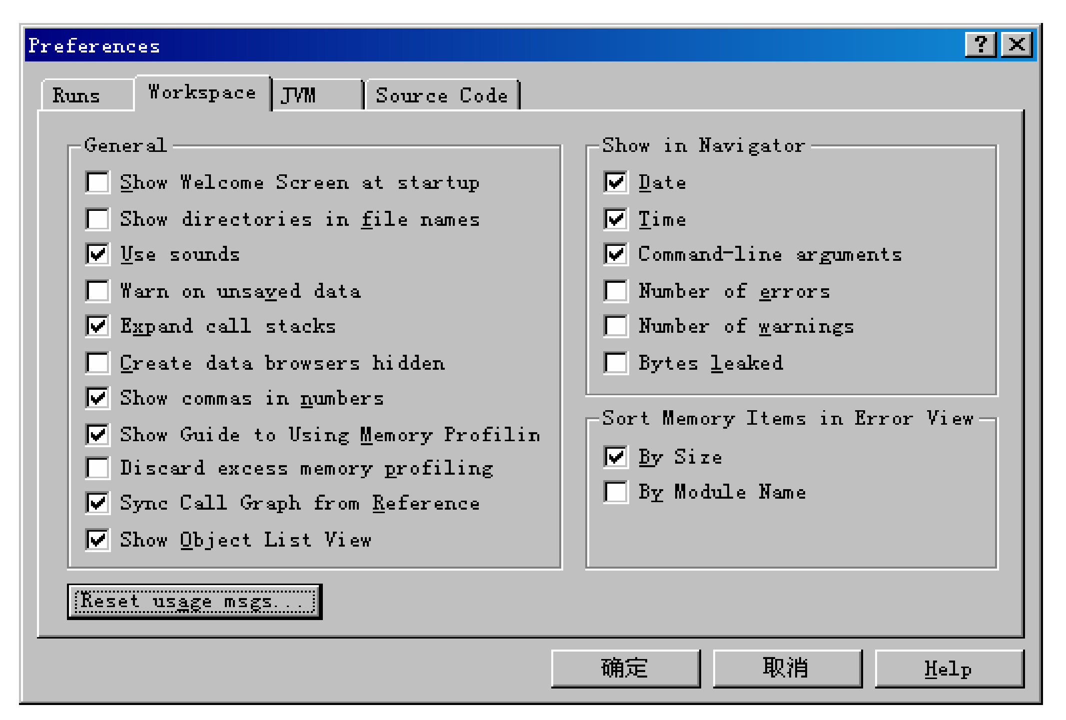
Show LoadLibrary instrumention progress：对 VC++、VB native-code 程序测试时，当工具文件需要调用工具列表时，是否显示工具对话框。

Confirm run concellation：当每次选择 file——Concel run 时，是否显示证实消息对话框。

Create automatic merge：创建自动的合并。
Use defaule filter set：使用缺省的过滤器设置。
Use case sensitive path name：区分大小写路径名。
Break on warnings in addition to error：当错误增加时中断。
Use the following debugger:使用下列调试器。

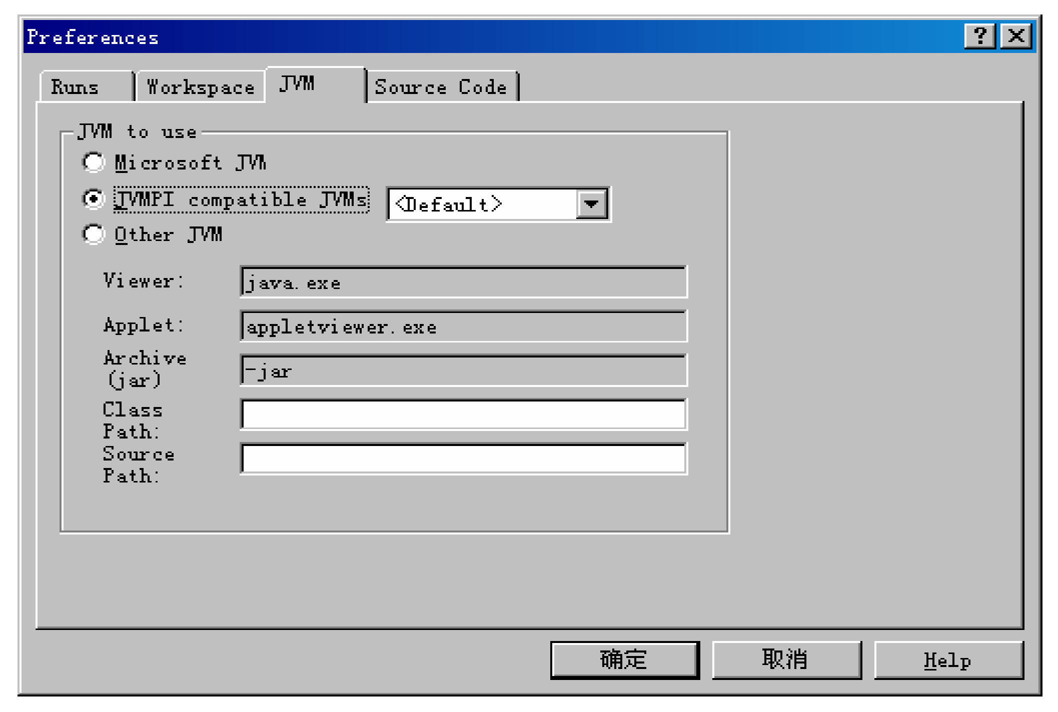


2)Workspace 选项卡：
Show Welcome Screen at startup：当每次打开该工具时，是否显示欢迎对话框。
Show directories in file names：当文件名显示时。在可能的情况下是否显示文件路径。Use sounds：是否有声音。
Warn on unsaved data：当关闭了一个为保存数据的运行时，或退出未保存数据的运行时，是否显示警告消息对话框。
Expand call stacks：扩展调用堆栈的个数。
Create data browsers hidden：创建隐含的数据浏览器。
Show commas in numbers：在数字中显示逗号。
Show Guide to Using Memory Profiling：显示引导使用内存压型。
Discard excess memory profiling：放弃剩余的内存压型。
Sync Call Graph from Rrference：同步参考调用曲线图。
Show Object List View：显示对象视图列表。
Show in Navigator：选择在 Navigator 窗口中是否显示日期、时间、命令行论述等信息。
Sort Memory Items in Error View：选择在错误视图中是否显示的内存种类信息。



3) JVM 选项卡

此选项卡是在测试 java 程序时，个性化 java 虚拟机是使用。表明使用什么虚拟机。



4) Source Code 选项卡：

Show C++ class names：显示 C++的类名。

Show C++ argument lists：显示 C++讨论列表。

Confirm recently changed source：确定最近源代码的改变。

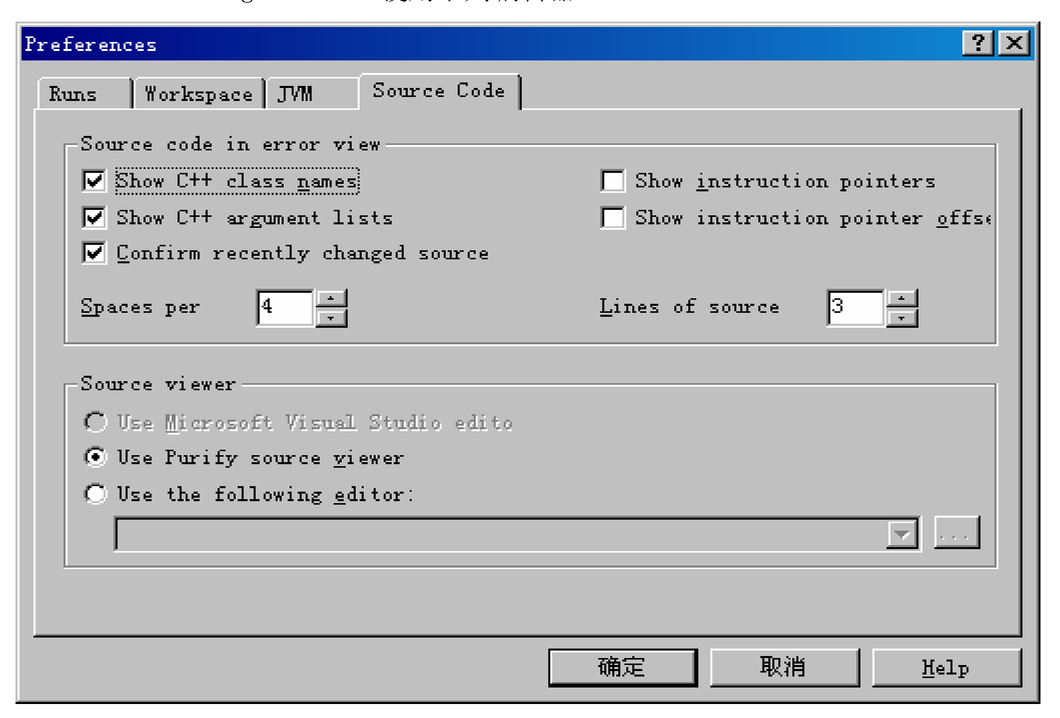
Show instruction pointers：显示指令指针。

Show instruction pointers offset：显示指令指针分支。

Spaces per：每间隔。

Lines of source：源代码的线程。

Use Microsoft Visual Studio editor：使用微软开发编辑器。
Use Purify source viewer：使用 Purify 源代码阅读器。
Use the following editor：使用下列编辑器。



2.2.3 View 当中的 Create Filter：

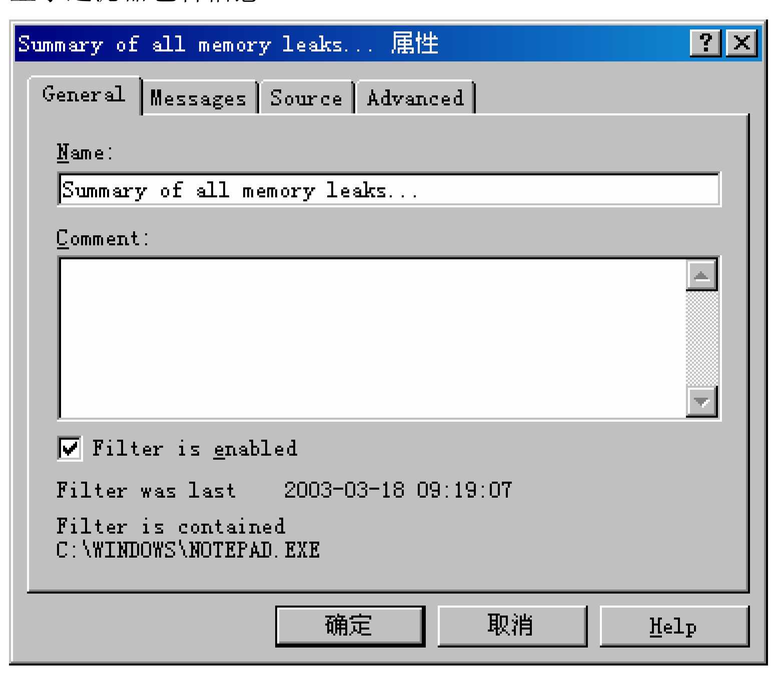
1) General 选项卡

定义过滤器的名称及注释。

设置过滤器是否可用。

显示过滤器的最后使用时间。

显示过滤器包含信息。

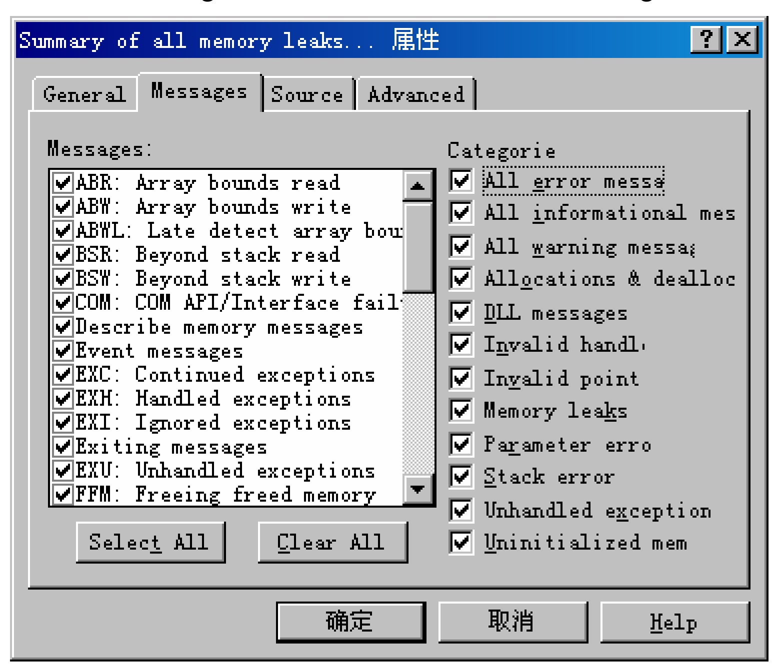


2)Messages 选项卡：

Categorie 种类：

All error messages	所有错误信息。
All informational messages	所有报告的信息。
All warning messages	所有警告的信息。
Allocations & deallocation	存储单元分配，地址分配和储存单元分配。
Dll messages	动态连接库信息。
Invalid handle	无效，非法的句柄。
Invalid pointer	无效，非法的指针。
Memory leaks	内存泄露。
Parameter error	参数错误。
Stack error	堆错误。
Unhandled exception	未曾用到的。
Uninitialized Memory Read (UMR).	未初始化内存阅读。

可直接在 Messages 栏选择需要的信息，也可在 Categorie 栏按种类选择所需要的信息。



3)Source 选项卡：

The messages this filter affects Function：这个过滤器所影响的函数。

Match if function is top function in call：如果函数是顶层调用函数时匹配。

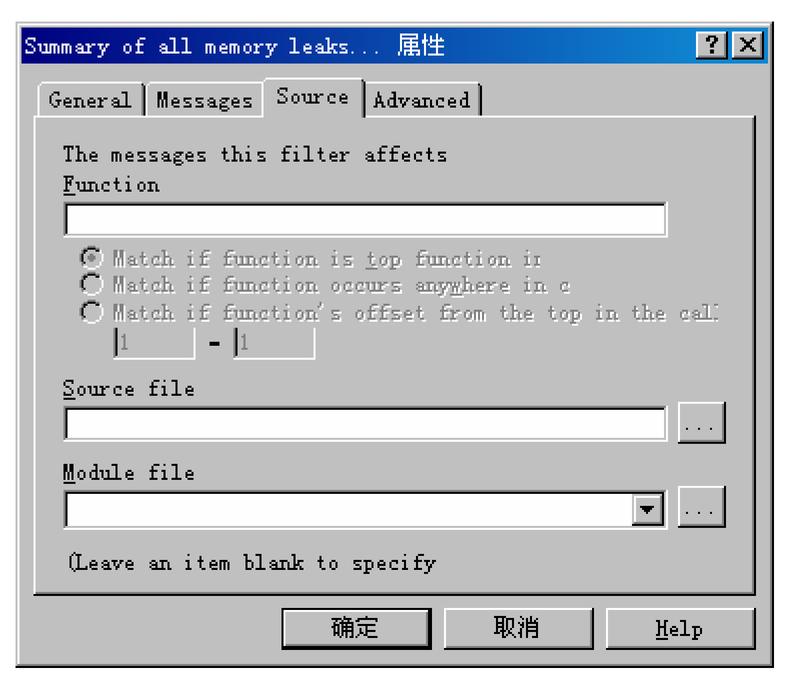
Match if function occurs anywhere in call：如果函数在任何地方被调用时匹配。

Match if function's offset from the top in the call：如果函数的误差来自顶层调用时匹配。

Source file：源代码文件。

Module file：模块文件。

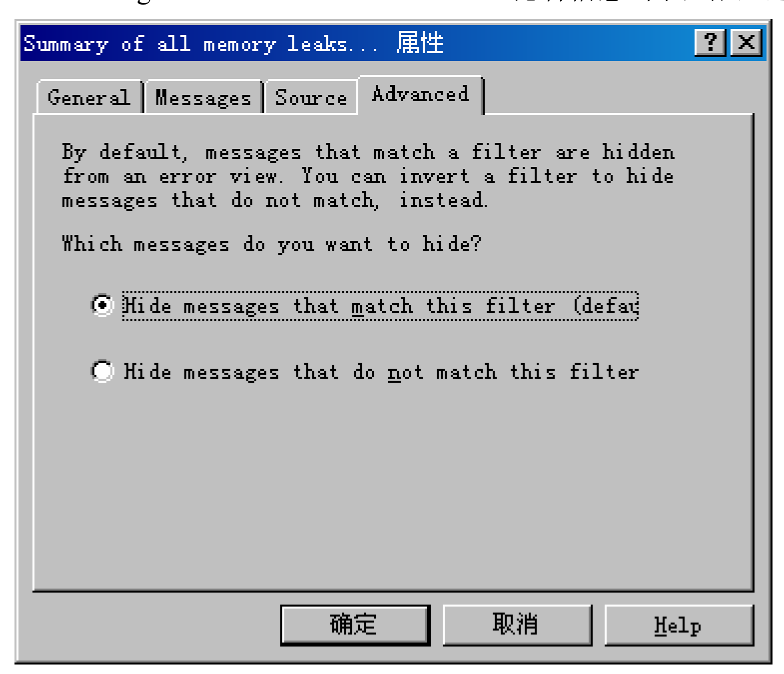
Leave an item blank to specify：允许项目在清单中空白。



4)Advanced 选项卡：

Hide messages that match this filter(default)：隐含信息当匹配此过滤器时。

Hide messages that do not match this filter：隐含信息当不匹配此过滤器时。



2.3 测试信息说明：

2.3.1 信息色彩:

红色：内存块没有被分配和初始化

蓝色：内存块已经被分配并且初始化了

黄色：内存块已经被分配并且没有初始化

2.3.2 名称缩写：

下列情况可引起内存的不可读或不可写:

Array Bounds Read (ABR)：数组越界。

Beyond Stack Read (BSR)：堆栈越界。

Free Memory Read (FMR)：空闲内存阅读。

Invalid Pointer Read (IPR)：非法指针阅读。

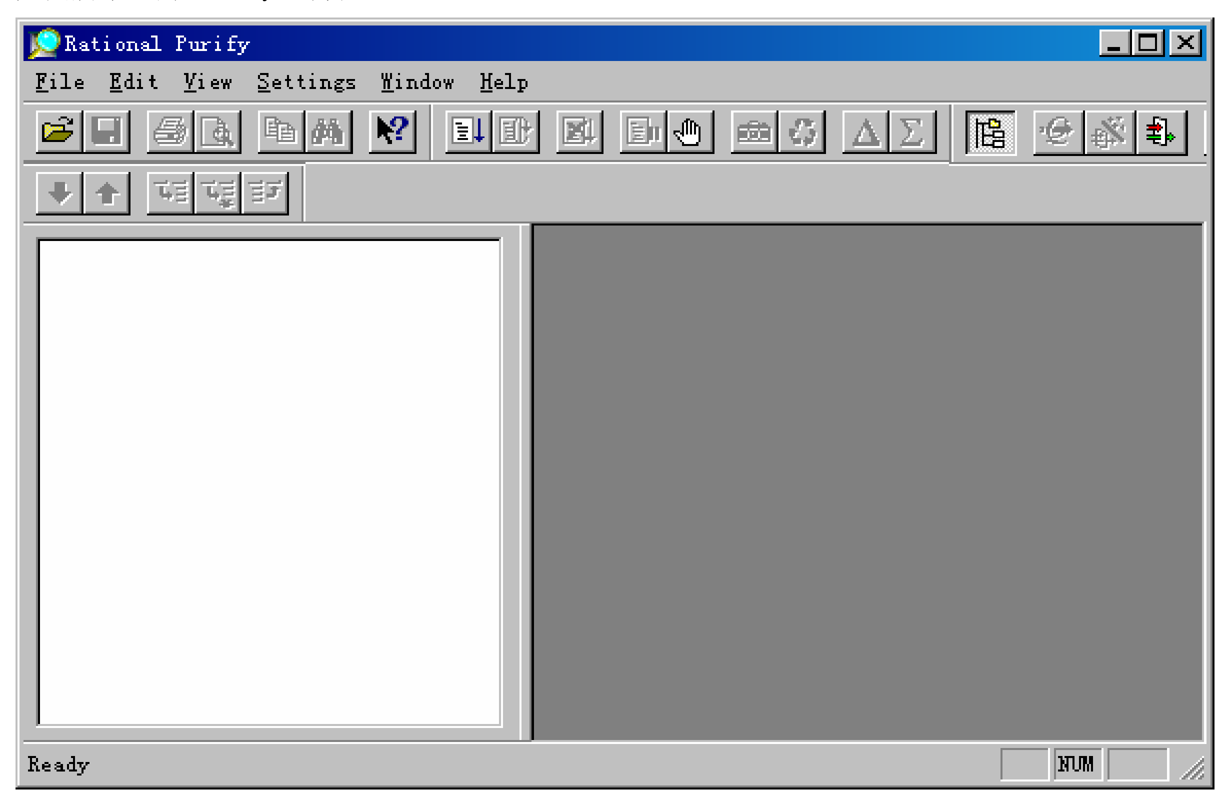
Null Pointer Read (NPR)：空指针阅读。

Uninitialized Memory Read (UMR)：未初始化内存阅读。

第三章 实用举例

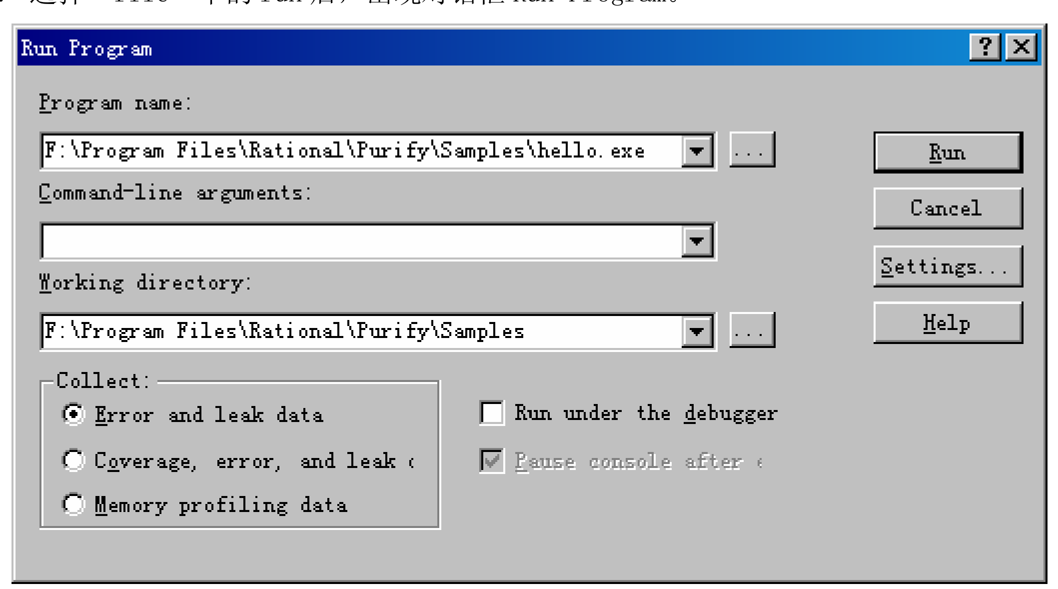
所选程序为 Java 语言开发。

第一步：从 windows“开始”菜单的“程序”中选择“Rational Suite Enteprise”选中“Purify”，如图所示，为 Purify 主界面。



第二步：测试被测程序

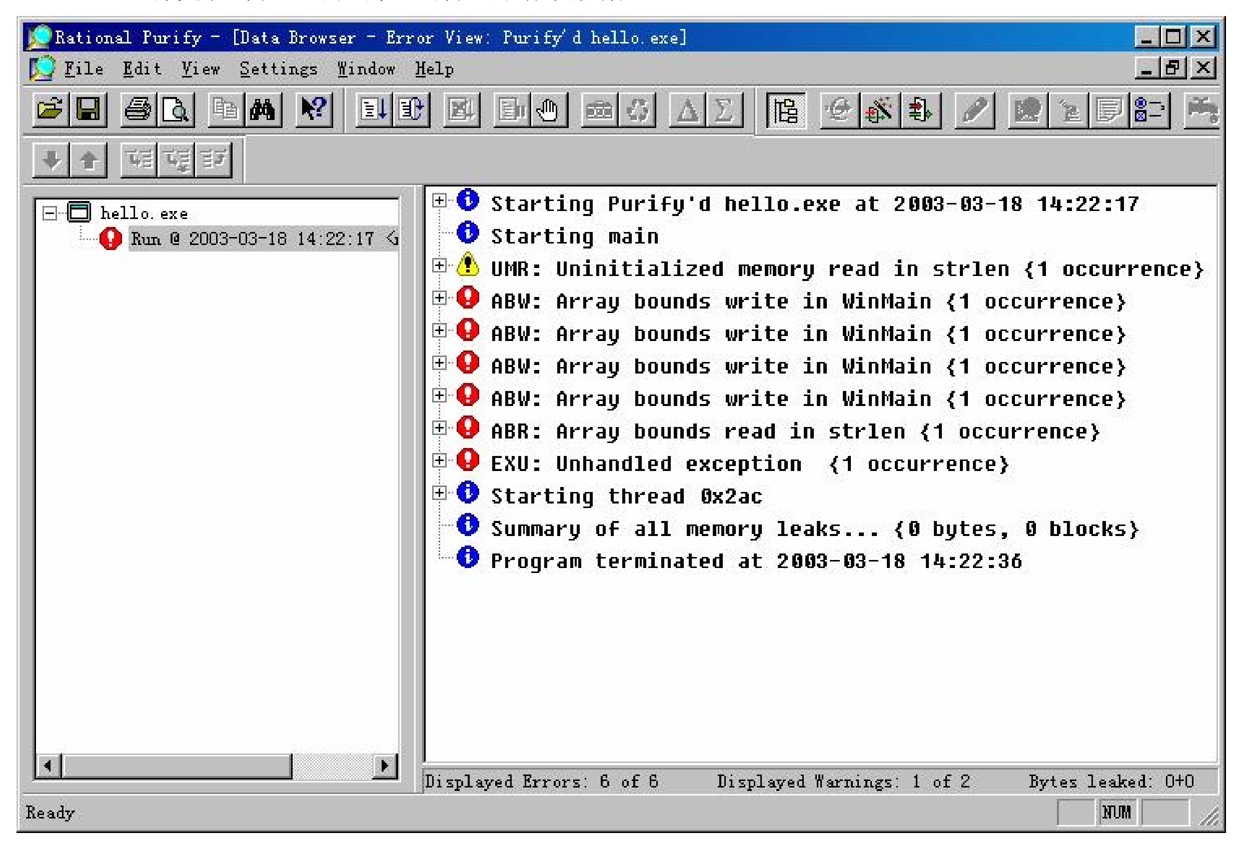
1. 选择“file”中的 run 后，出现对话框 Run Program。



2. 选择 Program name 中选择被测对象的路径后，点击 Run，运行程序。
   运行前可选择工作目录，以及输入命令行参数。
   并可在收集项中选择所要收集的信息类别。
   可选择是否在调试器下运行。



3. 运行完程序后，会出现运行后的结果数据。



通过此窗口，可以看到在程序运行期间内存的泄漏情况。

第三行标注未初始化内存阅读。

第四行至第七行标注数组越界导致内存不可写。

第八行标注数组越界导致内存不可读。

第九行标注未经处理的例外。

4. 双击 Data Browser 窗口中的任何一个错误或提示前面的“＋”号，均可看到该错误的详细信息。如果被测程序包含源代码，则在该错误的详细信息中列出错误的代码行并解释所造成的错误。

```
UMR: Uninitialized memory read in strlen {1 occurrence}
ABR: Array bounds read in strlen {1 occurrence}
    Reading 11 bytes from 0x003e45e8 (1 byte at 0x003e45f2 illegal)
    Address 0x003e45e8 is argument #1 of strlen
    Address 0x003e45e8 is at the beginning of a 10 byte block
    Address 0x003e45e8 points to a malloc'd block in heap 0x003e0000
    Thread ID: 0x56c
    Error location
        strlen          [strlen.obj]
        WinMain         [hello.c:35]
                for (i = 0; string1[i] != '\0'; i++) {
                    string2[i] = string1[i];          // ABW's generated on this line.
                }
                length = strlen(string2);             // ABR generated on this line.

                MessageBox(NULL, "        Hello, Windows", "The Windows Hello Dialog", MB_OK | MB_ICONINF

        WinMainCRTStartup [wincrt0.obj]
    Allocation location
        malloc          [malloc.obj]
        WinMain         [hello.c:28]
            {
                int i, length;
                char *string1 = "Hello, Windows";
                char *string2 = malloc(10);

                length = strlen(string2);             // UMR because string2 is not initialized.

        WinMainCRTStartup [wincrt0.obj]
Summary of all memory leaks... {682 bytes, 8 blocks}
Exiting with code 0 (0x00000000)
```

5. 保存测试信息，则将在与被测程序同一目录下生成一个.pfy 的文件，里面保存了 Data Browser 窗口的数据，以便进行数据共享。
6. 不论是否选择保存，在被测程序目录下都会生成一个文本文件，形成测试日志。